NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

NPSNET VEHICLE DATABASE:
AN OBJECT-ORIENTED DATABASE
IN A REAL-TIME VEHICLE SIMULATION

by

Susan C. Borden Davis

June, 1996

Thesis Advisor: Michael J. Zyda
Co-Advisor: David R. Pratt

Approved for public release; distribution is unlimited.

DTIC QUALITY INSPECTED 3

19961024 036

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY (Leave blank)	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	June 1996	Master's Thesis

4. TITLE AND SUBTITLE
NPSNET Vehicle Database: An Object-Oriented Database in a Real-Time Vehicle Simulation

5. FUNDING NUMBERS

6. AUTHOR(S)
Susan C. Borden Davis

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

The Naval Postgraduate School has actively explored the design and implementation of NPSNET, a real-time three-dimensional simulator on low-cost, readily accessible workstations. NPSNET involves a tremendous amount of interaction between vehicle, terrain, obstacle and ordnance objects in a dynamic simulation system. There exists a need for an organized, efficient storage structure that allows real-time retrieval of objects and their interactive relationships.

This work concentrates on selection and design of a vehicle database model to maximize storage and real-time retrieval of data for the NPSNET visual simulator. The results of this effort can be applied to the overall system, NPSNET, in a distributed database management system.

14. SUBJECT TERMS	15. NUMBER OF PAGES
database model, NPSNET, object-oriented	80
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED	UNCLASSIFIED	UL

NSN 7540-01-280-5500

Standard Form 298 (Rev 2-89)
Prescribed by ANSI Std 239-18

Approved for public release; distribution is unlimited

NPSNET VEHICLE DATABASE:
AN OBJECT-ORIENTED DATABASE
IN A REAL-TIME VEHICLE SIMULATION

Susan C. Borden Davis
Lieutenant Commander, United States Navy
B.S., United States Naval Academy, 1984

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
June 1996

Author: Susan C. Borden Davis
Susan C. Borden Davis

Approved by:
Michael J. Zyda, Thesis Advisor

David R. Pratt, Co-Advisor

Ted Lewis, Chairman,
Department of Computer Science

ABSTRACT

The Naval Postgraduate School has actively explored the design and implementation of NPSNET, a real-time three-dimensional simulator on low-cost, readily accessible workstations. NPSNET involves a tremendous amount of interaction between vehicle, terrain, obstacle and ordnance objects in a dynamic simulation system. There exists a need for an organized, efficient storage structure that allows real-time retrieval of objects and their interactive relationships.

This work concentrates on selection and design of a vehicle database model to maximize storage and real-time retrieval of data for the NPSNET visual simulator. The results of this effort can be applied to the overall system, NPSNET, in a distributed database management system.

TABLE OF CONTENTS

I. INTRODUCTION..... 1

A. BACKGROUND..... 1

1. NPSNET..... 1

2. NPSNET Vehicles..... 2

B. PURPOSE AND GOALS OF WORK..... 4

C. BREAKDOWN OF WORK..... 5

II. NPSNET VEHICLE INTERACTION EXAMPLES..... 7

A. INTERACTION WITH TERRAIN..... 7

B. INTERACTION WITH OBSTACLES AND VEHICLES... 8

1. Interaction with Obstacles..... 9

2. Interaction with Vehicles..... 10

C. INTERACTION WITH ORDNANCE..... 10

III. DATABASE EVOLUTION..... 13

A. HISTORICAL PERSPECTIVE..... 13

B. TRADITIONAL DATA MODELS..... 14

1. Hierarchical Data Model..... 15

2. Network Data Model..... 17

3. Relational Data Model..... 18

C. OBJECT DATA MODEL..... 21

IV. PROPOSED NPSNET VEHICLE DATABASE..... 27

A. NPSNET_V: DATA MODEL OVERVIEW..... 27

B. NPSNET_V: DATA MODEL DESIGN..... 28

1. Identify the Basis..... 30

2. Define the Requirements..... 31

3. Identify the Data Items..... 33

4. Separate the Data Members from the Classes..... 34

5. Build Data Member Dictionary..... 37

6. Gather Data Members into Classes..... 38

7. Identify the Key Members..... 44

8. Identify Class Relationships..... 46

9. Identify the Methods..... 47

10. Make the Class Persistent..... 48

C. NPSNET_V: DATA MODEL IMPLEMENTATION..... 49

1. Define NPSNET_V Database..... 50

2. Bring NPSNET_V Together..... 51

V. CONCLUSION..... 53

A. RESULTS..... 53

B. RECOMMENDATIONS..... 54

APPENDIX..... 57

GLOSSARY..... 61

REFERENCES..... 65

INITIAL DISTRIBUTION LIST..... 67

LIST OF FIGURES

Figure 3.1 - A Hierarchical Database..... 16

Figure 3.2 - A Network Database..... 17

Figure 3.3 - A Relational Database..... 19

Figure 3.4 - A Many-to-Many Relationship..... 20

Figure 3.5 - A Class Hierarchy..... 23

Figure 3.6 - A Land_Vehicle Class Hierarchy..... 24

Figure 4.1 - Ten Basic Object Database Design Steps... 29

Figure 4.2 - Object Database Design Step 1: Identify the Basis..... 31

Figure 4.3 - Object Database Design Step 2: Define the Requirements..... 32

Figure 4.4 - Object Database Design Step 3: Identify the Data Items..... 34

Figure 4.5 - Refined List of NPSNET_V Data Items..... 35

Figure 4.6 - Object Database Design Step 4: Separate the Data Members from the Classes..... 36

Figure 4.7 - Object Database Design Step 6: Gather Data Members into Classes..... 39

Figure 4.8 - The **MK_46** Class..... 41

Figure 4.9 - The **SH_60B** Class..... 42

Figure 4.10 - The **DDG_51** Class..... 43

Figure 4.11 - The **Assignment** Class..... 44

Figure 4.12 - Object Database Design Step 9: Identify the Methods..... 47

Figure 4.13 - Expanded list of NPSNET_V Class Methods.. 48

ACKNOWLEDGEMENT

Many thanks to Drs. Zyda and Pratt for their most excellent NPSNET adventure, and especially to my family for their tenacious support, and to my mother, whose spirit guided me through to completion.

I. INTRODUCTION

This work covers the selection and design of a database model to store and retrieve data for NPSNET, a real-time vehicle and battle simulator. NPSNET is a multiple year effort started in early 1990 as a newer version of the older, more expensive SIMNET project. It explores the SIMNET domain using readily available IRIS graphics workstations rather than platform specific nodes, and provides a real-time interface with the user for virtual world exploration and experimentation [Osbo91]. Maximizing storage and real-time data retrieval for the simulator can lead to development of a viable distributed database system.

A. BACKGROUND

1. NPSNET

NPSNET is a real-time vehicle and battle simulator capable of displaying vehicle movement over the ground, in the air or through water. While NPSNET supports a full complement of land, air and sea vehicles, it also provides the mediums through which these vehicles move. It models terrain features such as soil type, water masses, vegetation and elevation, and can effect environmental conditions including coastal fog or urban haze. NPSNET incorporates the cultural features of roads, buildings, signs and other fixed objects to fully immerse the user into its simulated world.

The NPSNET user selects a vehicle from a field of realistic craft, fanciful flying cows, turkeys or tomato tanks, and controls its movement via a six degree of freedom spaceball or button/dialbox associated with a command and control screen. Up to 500 vehicles may interact in the virtual world at a given time, including autonomously scripted vehicles or vehicles controlled by other users networked to the system. As NPSNET constructs a three-dimensional virtual world in which to move and interact, it pursues training, planning, gaming and other purposes where the introduction of a physical player may be too hazardous, too expensive or too frivolous to tolerate [Zyd92].

2. NPSNET Vehicles

NPSNET engages to totally immerse the user in its real-time vehicle combat simulation. The graphics representing the virtual world respond to the actions and movements of the user and convince the user that they truly are within the environment that the simulator represents [Zyda93]. With this long-term goal in mind, NPSNET supports a full complement of land, air and sea vehicles capable of simulated movement over ground, in air or through water. It models the mediums through which these vehicles move with terrain features, and incorporates the fixed objects around which these vehicles move with cultural features, or obstacles. NPSNET represents vehicle armament with ordnance features, adding accurate aural cues to provide feedback

about the user's vehicle, environment and actions taking place.

The NPSNET user chooses a vehicle by making a mouse selection on a command and control screen. Each vehicle is actually a low resolution indicator of a player on the terrain. Its three-dimensional icon displays a minimum level of graphical detail with which the NPSNET user is willing to live [Zyd92].

An NPSNET vehicle operates in an environment of gridsquares, with up to 500 vehicles interacting in the virtual world at a given time. Interactions may involve vehicles controlled by prewritten script, vehicles driven interactively from other workstations in the network and autonomous vehicles introduced into the system via a programmable network "harness" process [Zyda92]. A two-dimensional map on the command and control screen displays the position and tracking of all players attached within a gridsquare, as well as the direction and scope of the user's vehicle and the position and movement of the other vehicles.

The user accesses additional data through a window at the top of the viewing screen. For instance, the window may display the speed, direction and attitude of the engaged vehicle, its remaining ordnance and current fuel level. The user should ultimately be able to access all data pertinent to any vehicle or related object within the virtual world, expanding the tutorial capacity of the simulation.

Players control their chosen vehicle using a variety of interface devices, including a keyboard, a button/dialbox and a six degree of freedom spaceball. The latter is an extremely versatile device for control of three-dimensional movement, facilitating quick user action.

Real-time NPSNET representation relies on rapid event detection and reaction, including reacting to user input, following terrain contours, and responding to changes in the three-dimensional world [Osbo91]. Real-time combat simulation also relies on rapid detection of, and response to, interaction between land, air and sea forces. Chapter II of this work covers NPSNET vehicle interactions, along with a discussion of collision detection and response.

B. PURPOSE AND GOALS OF WORK

Current NPSNET data storage and retrieval mechanisms are slow, cumbersome to maintain and complex to enhance. As NPSNET proceeds in the direction of object-oriented modeling and real-time scene management over a multiple workstation network, its data management system must also progress. NPSNET requires an updated database facility to support the demands involved in real-time vehicle combat simulation. Dynamic terrain and ballistic motion issues, intelligent autonomous agents, a three-dimensional sound system, and other future projects will require speed and accuracy in documenting world state events, with built-in concurrency control to manage user interface across a distributed network.

This work identifies, develops and explores the design of a data model to enhance the existing NPSNET data management system. It compares traditional database technology and proposes a viable alternative to meet the increasing data management needs of the NPSNET simulator, building upon this selection to design a model of the proposed database. This work also uncovers areas for future research and development.

C. BREAKDOWN OF WORK

Chapter II covers various types of NPSNET vehicle interaction. These include examples of terrain traversal, collision with fixed and moving objects, and changes rendered by ordnance impact.

Chapter III provides a detailed description of traditional and available data models, their primary areas of application, and a discussion of their utility in the NPSNET environment. Italicized terms are in the Glossary.

Chapter IV presents a description of the proposed database mechanism for use with NPSNET, with a discussion of its selection and design. It references sample data lattices, found in the Appendix. This chapter also discusses implementation issues affecting the proposed data model within NPSNET.

Chapter V concludes this work. It reviews the proposed NPSNET Vehicle Database and recommends areas for future consideration.

II. NPSNET VEHICLE INTERACTION EXAMPLES

The NPSNET visual combat simulator models land, air and sea vehicle operation. The virtual world in which the vehicles operate includes terrain features, cultural features representing fixed objects, or obstacles, and weaponry, or ordnance, features. All simulated objects within NPSNET fall under the umbrella of a vehicle, terrain, obstacle or ordnance class, respectively.

NPSNET achieves virtual world battlefield exploration and experimentation using interactions between objects of the vehicle, terrain, obstacle and ordnance classes. A discussion of these object classes, with examples of their interaction in a real-time interface with the user, follows.

A. INTERACTION WITH TERRAIN

The first step in virtual world development is to obtain data that represents the world the system will model. A three-dimensional visual simulation commonly starts with a large two-dimensional grid of elevation data and converts it into a three-dimensional terrain carpet [Zyd92]. NPSNET divides the terrain data of the original SIMNET database into gridsquares designed to accurately represent actual terrain features such as soil type, water masses, vegetation and elevation. Additionally, it effects environmental conditions including coastal fog or urban haze.

Vehicle interaction with the environment will further evolve terrain representation into a more dynamic facsimile. The concept of dynamic terrain recognizes that simulated objects can affect the environment and that these effects must be recognizable throughout the simulation. If, for example, a vehicle knocks down a tree, dynamic terrain will represent the tree throughout the network as knocked down, and from that point forward, users joining the system must view the virtual world with that tree knocked down, with the change recorded so that the episode can resume even if a hiatus occurs [Zyda93].

Ordnance can also interact with the terrain. An artillery round fired from a vehicle and creating a crater upon hitting the ground is another example in which simulated interaction imposes lasting effects upon the environment.

The database must save dynamic terrain modifications and relay them across the NPSNET system so that all users view the same state of the world model. The credibility of the system, otherwise known as immersion, is contingent upon successful real-time storage and retrieval of this data.

B. INTERACTION WITH OBSTACLES AND VEHICLES

In the first NPSNET incarnation, vehicles could pass through fixed and moving objects, decreasing the immersion of the user in the virtual world. NPSNET now integrates collision detection into its overall system, enhancing

realism and increasing system data needs as the computer constantly searches the world model to determine if a vehicle is sharing its space with another object. NPSNET further maintains its usefulness with real-time collision response that includes an assessment and report of damages that colliding objects suffer. An accurate database and efficient management system will facilitate processing of simulated vehicle interaction.

1. Interaction with Obstacles

NPSNET incorporates the cultural features of roads, buildings, signs and other fixed objects to more fully immerse the user into its simulated world. It documents instances of vehicle contact with such obstacles and records system responses for network distribution so that all users interact with the same state of the world model.

As a vehicle proceeds through NPSNET, an algorithm updates its position and checks for an object collision. The algorithm maintains real-time simulation by limiting the scope of the collision detection to those obstacles attached within the current gridsquare radius of vehicle movement [Osbo91].

An example of vehicle contact with a cultural feature is the positioning of a ship alongside a pier. Collision detection identifies the immediate proximity of the vessel to the structure and collision response relays the success of the simulated docking maneuver to the user.

A vehicle collision with a fixed obstacle such as a watertower affects the simulated environment as in the dynamic terrain example given previously. A parked vehicle also affects the environment when it ceases vehicle movement through a terrain gridsquare and becomes an obstacle attached to the terrain by parking within that gridsquare.

2. Interaction with Vehicles

The potential exists for a user's vehicle to collide with any of the other NPSNET land, air or sea vehicles. A collision check for moving objects first limits the scope of the collision detection range to identify only probable collision participants, then determines if a collision has occurred before determining the actual point of collision [Osbo91]. Next, NPSNET runs a response function to display the extent of damage to the vehicles involved.

A common example of a vehicle collision is a crash between two land vehicles, such as a jeep running into a tank. Other vehicle interaction examples include a fighter jet landing on an aircraft carrier or a cargo truck driving onto a railcar platform. In these instances, NPSNET will reflect both the individual characteristics that each vehicle retains and the resulting damage or composite state of the vehicles involved.

C. INTERACTION WITH ORDNANCE

NPSNET vehicles also interact with their own weaponry and the armament of other vehicles. NPSNET will note the

impact of ordnance upon a vehicle, such as missile contact with a personnel carrier, detecting it similarly to a collision between vehicles. Collision response alters the world model state to reflect casualties back to networked workstations.

NPSNET attempts to regulate and monitor the use or release of vehicle ordnance, updating statistical characteristics accordingly. Weapons realistically deploy ammunition. A submarine, for example, cannot launch a missile that it does not have, nor can a machine gun continue to fire after its rounds are spent. Ordnance behavior incorporates independent propulsion and ballistic motion issues.

Dynamic terrain comes into play with ordnance interaction. As terrain state changes, NPSNET will capture and record any alterations, noting obstacle modifications if an interaction with ordnance changes cultural features to impose a lasting effect upon the environment. A crater left by an artillery round highlights a good example of an altered terrain state.

NPSNET stresses truthfulness in its simulation of object interaction. To teach a user incorrectly in a vehicle combat simulator puts that individual in danger and could perhaps lead to a lethal situation when the user encounters reality, losing the usefulness of the simulation [Zyda93]. Accurate vehicle interaction relies heavily on an effective data

model with efficient storage and retrieval mechanisms. A discussion of certain traditional and available data models, their evolution, and their relevance in the NPSNET environment appears in the next chapter.

III. DATABASE EVOLUTION

Databases have evolved over the past two decades from rudimentary, ad hoc systems into central components of organizational information systems [Higa92]. In the relatively short time since commercial data management products first appeared, this area of computer research and development has become a primary field of fundamental and conceptual importance [Date81]. This chapter introduces *database management* and documents its progression through the three best known data paradigms, the *hierarchical*, *network* and *relational data models*. It concludes with a discussion of emerging trends in data modeling and data processing that are propelling the field of data management toward an *object-oriented* front, presenting a detailed description of the new and powerful *object data model*. Italicized terms found in this chapter are in the Glossary.

A. HISTORICAL PERSPECTIVE

A *database management system* entails a database and a set of programs that allow users to access and modify system data *files* [Date81]. A *data model* specifies a given database, and then refers to it in terms of the abstract and concrete features of the *types*, *aggregates* and *relationships* of data within that database. A *data language* provides access to the database as it specifies data processing, integrity and security requirements [Hsia91]. Together, a

data model, data language and database management system characterize the concept of database management.

In order to better understand our current needs and future direction in data management, we should first understand the history of database management, which traces the history of data processing itself [Stev92]. With the conception of information processing machines, programmers have faced the challenge to manage data and to store and organize it into formats allowing for rapid and flexible processing. While these basic information systems requirements remain, system architectures continue to change significantly as more powerful development tools produce increasingly complex applications [Andl92].

Early computing systems were proprietary, with database management ad hoc, customized for specific applications. As computer usage increased, there was a parallel rise in user demand for support of a wider variety of applications, and the mid 1960s saw the first generation of database management systems [Andl92]. The introduction of standard, general-purpose systems freed developers from creating new database management software for each new application, but the use of such systems mandated the definition of certain data models that system designers would use [Stev92].

B. TRADITIONAL DATA MODELS

Data management for the mainframes of the 1960s conserved memory space and processing load, organizing data

along clearly defined access paths, evolving into the hierarchical and network data models [Andl92]. The minicomputers of the late 1970s and the early 1980s brought the need for interactive, flexible data management, and the relational data model promised to meet these new requirements. What follows is a brief description of each of these three traditional data models.

1. Hierarchical Data Model

The hierarchical data model was one of the first formal data management approaches, supporting hierarchical organizations that exist in the real world by representing data in an inverted tree structure. A hierarchical data model accesses data from the top to the bottom of its structure in a series of nodes, similar to branches in a tree, with embedded physical pointers in the data *records* to support interfile relationships.

The hierarchical format defines the concepts of a *parent* record, a *child* record, a record type, and parent-child relationships, observing properties as follows [Elma89]: (1) the hierarchy root does not participate as a child record type in any parent-child relationship; (2) every record type except the root does participate as a child record type, with only one distinct parent for each child; (3) any parent record type can participate as parent in any number (zero or more) of parent-child relationships; (4) a record type that does not participate as a parent

record type in any parent-child relationship is a leaf; and (5) if a record type participates as parent in more than one parent-child relationship, then its child record types are ordered, with the order displayed, by convention, from left to right in a hierarchical diagram.

Figure 3.1 illustrates the hierarchical relationships of an organizational database that has files of companies, divisions within each company, and departments within each division [Stev92]:

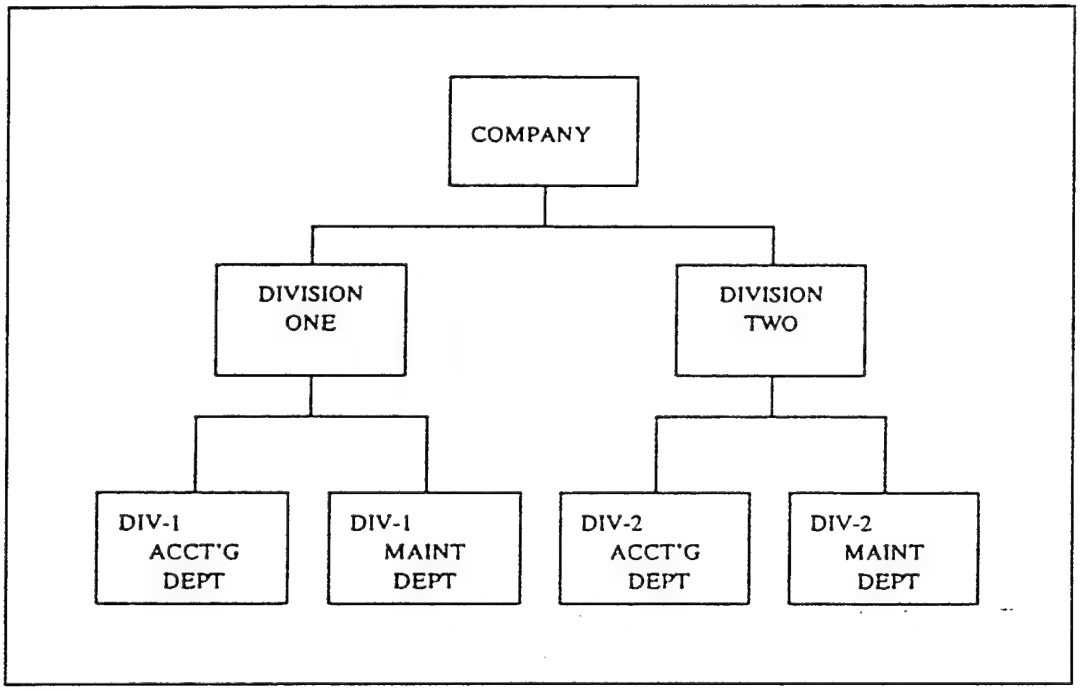


Figure 3.1 A Hierarchical Database

Some disadvantages result from the rigidity of the hierarchical structure. Modification of the database structure is a very complex task because a previously undefined, new access path may be complicated or even impossible to achieve [Andl92]. Additionally, the

hierarchical data model has no ability to represent records with multiple parent relationships, unlike the network data model.

2. Network Data Model

The network data model evolved from the need to easily depict non-hierarchical relationships. If the organization of Figure 3.1 were to allow one department to support multiple divisions, the network data model would be able to directly portray the ensuing parent-child relationships. Figure 3.2 [Stev92] shows a network representation of a department record with multiple parents:

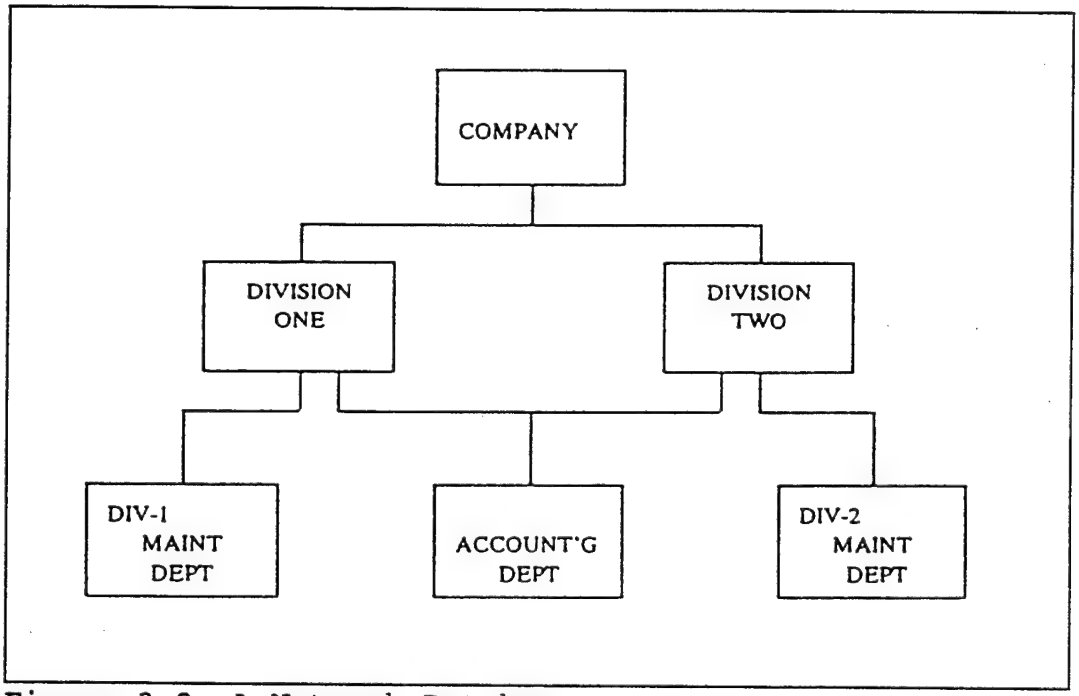


Figure 3.2 A Network Database

While a network continues to organize data in an inverted tree, it is a more general structure than a hierarchy. A given record occurrence in a network may have

any number of immediate superiors, modeling a many-to-many correspondence more directly than the hierarchical approach [Date81]. Predefined *pointers* link records, increasing flexibility over the hierarchical data model.

The disadvantage of the network data model is its reduction in high-access performance. The complexity of the network structure makes queries more complex and modification more complicated. While the network data model of data management is ideal for information systems departments to use for large-scale batch processing, it does not meet the need for interactive or ad hoc data management, as does the relational data model [Andl92].

3. Relational Data Model

The relational data model is the best known of the three traditional record-based data models [Hsia92]. The entire data file format of the relational database is visible to the user. Data in the relational data model is stored in tables consisting of columns and rows, with the rows representing data records and the columns representing the fields in a record. The database uses a *primary key* data *element* to uniquely identify each record.

Figure 3.3 [Stev92] shows a typical relational database in which the primary key of one record is a *secondary key* for another record. This allows an application to retrieve records of one type based on the specified primary key of another type.

In this case, the division identification number is the primary key of the division record. Because each department record contains the identification of the division to which it belongs, the division identification number is the secondary key of the department record.

Since many departments can belong to one division, the application can use this secondary key to retrieve a list of which departments belong to a particular division. The arrow icon, with its single and double arrowheads, represents a one-to-many relationship linking the two records.

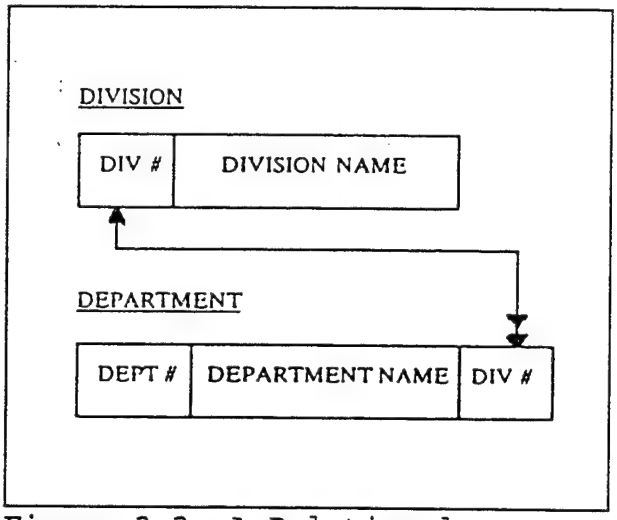


Figure 3.3 A Relational Database

To support a many-to-many relationship, the relational data model uses a connector file to link record definitions. A concatenation of primary keys from the linked records forms the primary key to the connector file, while either data element by itself forms a secondary key [Stev92]. The connector file may also contain data items that are unique to the connection itself.

In Figure 3.4, the division and department identification numbers together form the primary key of the connector file. Individually, each is a secondary key of

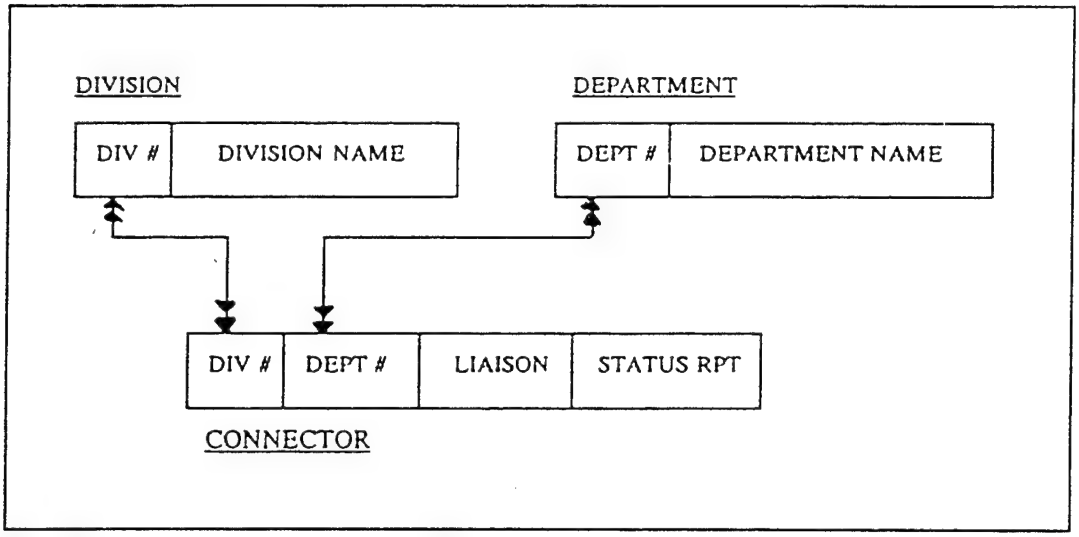


Figure 3.4 A Many-to-Many Relationship

the connector file, as well as the primary key of their respective file.

The relational data model uses actual data values in its records, rather than hidden pointers, to represent file relationships and locate data elements. This associative access is less vulnerable than navigating by pointer, for it eliminates the risk of broken pointer chains, increases data integrity, and facilitates recovery from system error.

A general-purpose relational database management system is relatively easy to design and develop, and can represent any interfile relationship found in other record-based data systems. The relational approach examines database file relationships to reorganize data elements in an effort to eliminate redundancy, internal file pointers, and repeating groups of data elements in the records. These points combine with its inherent strength to make the

relational data model the standard method for representing data in a database [Stev92].

The relational data model achieves its superior flexibility and maintainability at the cost of a higher level of processing to establish connections and access data. Its systems requirements for processor and memory are higher to achieve the same level of overall performance found in tree structures [Andl92].

The demand for more powerful and flexible distributed relational database management systems, when coupled with the desire to support variable-length data items, repeating groups, and *abstract data types*, leads us beyond the traditional data models to investigate a fourth kind of data model. This new paradigm, the object data model, is a merging of object-oriented programming and traditional database technology [Booc91].

C. OBJECT DATA MODEL

Design of advanced programming languages and environments provided the primary introduction of the object-oriented paradigm, which directly represents real-world objects by database objects [Bert91]. The pervasive growth of object-oriented programming technology makes it the chosen technique for software development in the 1990s.

Object-oriented programming derives its strength from four fundamental characteristics [Stev92]: *abstraction, encapsulation, inheritance* and *polymorphism*. Abstraction is

the ability to design new abstract data types for object representation. Encapsulation binds the behavior of an object to its data values in one logical unit. Inheritance allows new data types to derive, or inherit, behavior from old ones, and polymorphism customizes the behavior of a derived data type.

The merits of using an object-oriented approach to database management stem from its perceived power over traditional approaches and its inclusion of the robust constructs, functions and features that allow programmers to capture the organic elements of their application [Hsia91]. While conventional data models "scatter" information about an object over several records or files [Elma89], object-oriented programming promises a more natural relationship between information and its processing. Because object-oriented design can represent data in ways that traditional data models cannot, it needs a database model of its own. The object data model is the first attempt to marry a programming model with a database model [Stev92].

The object approach to data management efficiently provides a single, object-oriented language to define both the data and the user interface [Andl92]. The object data model describes the structure of its system *objects*, including their identity, their relationships to other objects, their attributes, and their operations [Rumb91]. Each object-oriented program is a cooperative collection of

these system objects, and each object represents an *instance* of some *class*. Object-oriented classes are members of a *class hierarchy*, whose inheritance relationships unite them [Booc91]. Classes define the attribute values that each object instance carries, and the operations which each object performs or undergoes [Rumb91].

Graphical representations of the object model, like the one in Figure 3.5, diagram object classes, arranging them into hierarchies that share common structure and behavior with which other classes can associate [Rumb91]. In a multilevel class hierarchy, the root of each class is an object and the instances of each class are the classes at

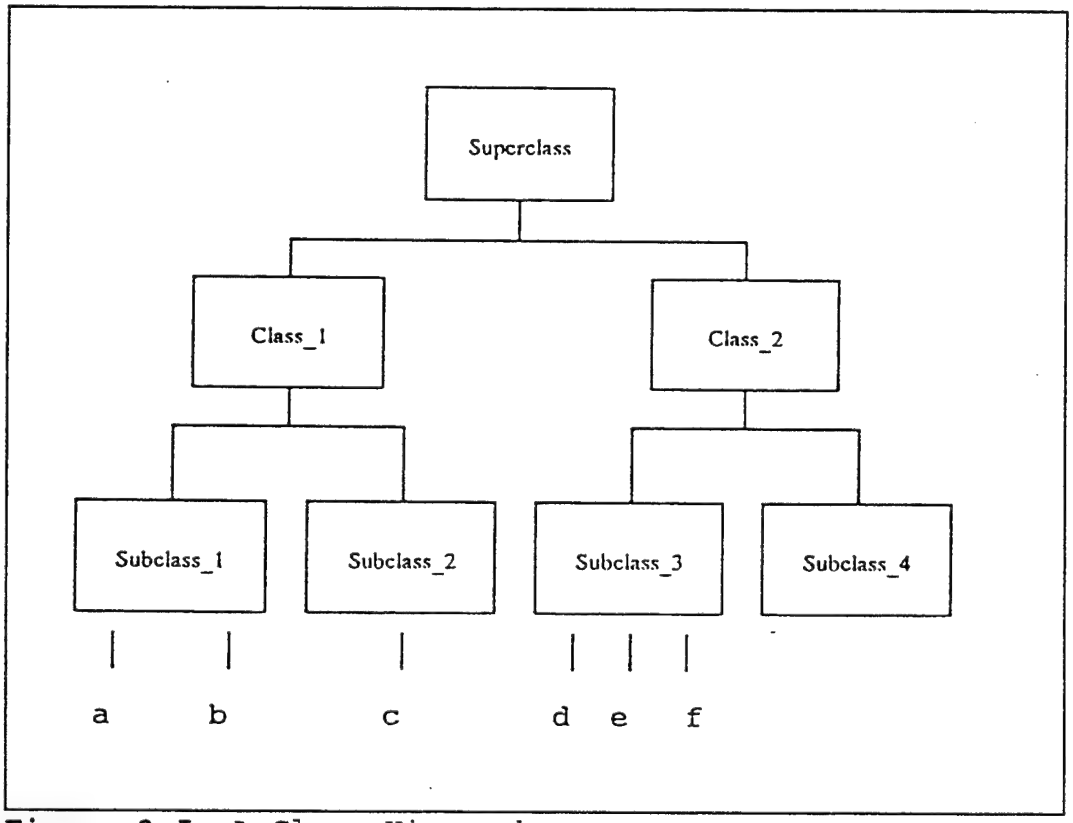


Figure 3.5 A Class Hierarchy

the next lower level. The lowest level of the generic class hierarchy leads to the objects that are instances of the lowest-level class that comprises the objects [Andl92]. Each lower level, or *subclass*, is derived from a *base class*, or *superclass*.

Figure 3.6 replaces generic class and object names to describe a "Land_Vehicle" object data model, where object classes such as "M1" and "M113" are at the lowest level of the "Tracked" class, which in turn is a subclass of the "Land_Vehicle" superclass. Each object instance of the tracked land vehicle "M1" is the lowest-level object in this hierarchy.

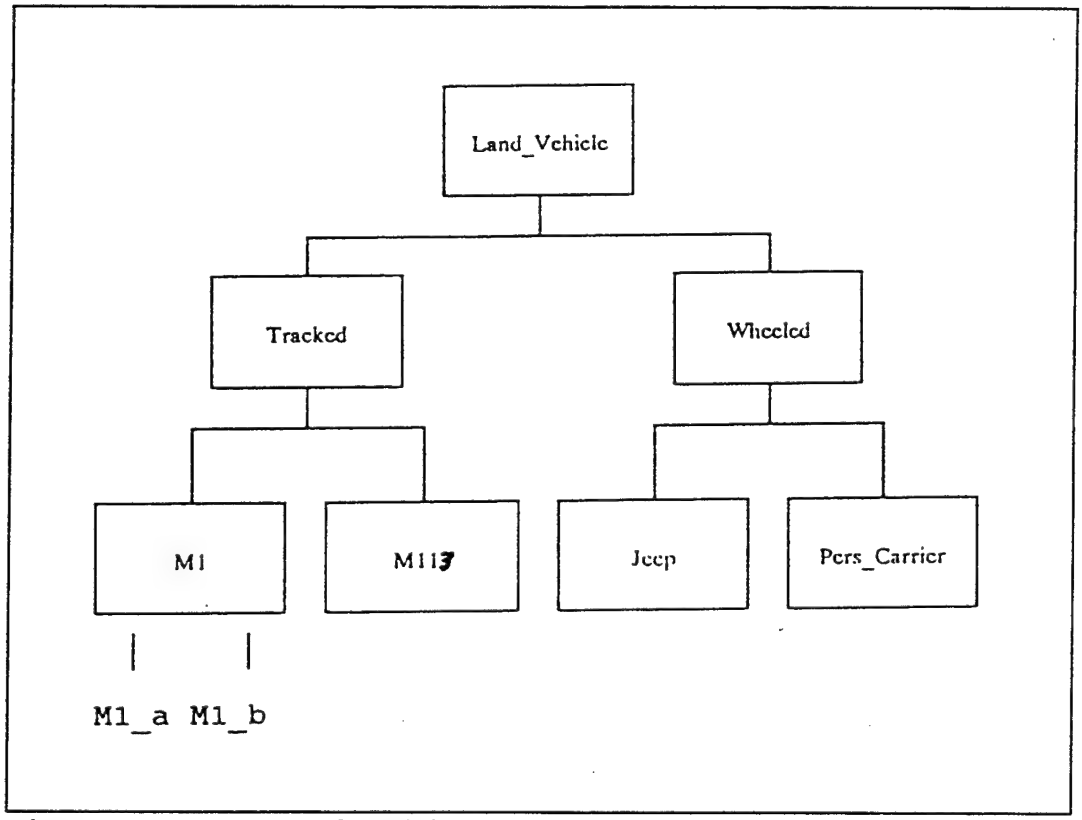


Figure 3.6 A Land_Vehicle Class Hierarchy

A comparison of available database technology to the needs of a virtual world system like NPSNET highlights some traditional data model shortfalls. Hierarchical and network databases are weak choices for NPSNET data management, as they rely on hidden pointers, are often complex and sometimes impossible to modify, and do not meet the need for interactive data management. Relational data models overcome these problems but their rules prohibit many of the data representations that an object paradigm can support.

A real-time vehicle combat simulator like NPSNET requires a data management system that can accommodate the vast array of graphic object imagery, movement and sound that combine to fully immerse the user in its world. The object data model is a viable option to represent NPSNET. The following chapter expands on the selection and design of the proposed NPSNET vehicle database, and discusses some implementation issues affecting the proposal.

IV. PROPOSED NPSNET VEHICLE DATABASE

NPSNET simulates a full complement of land, air and sea vehicles, their ordnance, their movement, and the mediums through which they move. An NPSNET storage and retrieval facility should support the varied demands involved in real-time vehicle combat simulation, including intelligent autonomous agents, a three dimensional sound system, dynamic terrain and ballistic motion issues, multiple workstations and built-in concurrency control. The progression of NPSNET toward object-oriented modeling and real-time scene management over a distributed network highlights its need for an updated data management system. NPSNET_V, a proposed NPSNET Vehicle Database, emulates the persistent features of a relational database while capitalizing on the object-oriented characteristics of abstraction, encapsulation, inheritance and polymorphism. This chapter discusses the selection and design of the NPSNET_V object data model and then explores its implementation.

A. NPSNET_V: DATA MODEL OVERVIEW

Comparison of the strengths and weaknesses of available data models indicates that NPSNET_V, a proposed NPSNET Vehicle Database, would benefit from a combinational design approach. Going beyond traditional data models, NPSNET_V experiments with object-oriented database design to meet the increasing data management requirements of NPSNET. It

adapts the relational data model to its NPSNET objects to combine the strengths of a relational model with the inherent capabilities of object-oriented design, defining related persistent classes and their behavior within the NPSNET vehicle combat simulator.

As an object data model, NPSNET_V can represent variable length data members, using abstraction to accommodate NPSNET simulations such as imagery, multimedia, geographic data and weather. Encapsulation allows NPSNET_V to bind data representation and behavior to more naturally represent the NPSNET modeled world. Through inheritance, NPSNET_V builds a hierarchy of derived classes, and it uses polymorphism to further customize the behavior of its derived data types.

Drawing strength from the relational model, NPSNET_V achieves persistence by defining a base class that participates in its own persistence, from which it can derive its other classes. NPSNET_V relates object locations and relationships by associating key data values with each object instance, and it maintains the integrity of those relationships to keep the database in a stable condition with respect to the real-time scene management needs of NPSNET over its distributed network [Stev92].

B. NPSNET_V: DATA MODEL DESIGN

Object database technology is still so new that no formal design methodology exists. While the basic concepts of objects, classes and inheritance seem widely understood,

common use of inheritance and composition to achieve a useful and robust class hierarchy is not yet well-defined [dePa91]. Instead, various guidelines for object data modeling present themselves to designers who favor an object-oriented approach over established, but less versatile, data models.

NPSNET_V Design

Step 1. Identify the basis.

Step 2. Define the database's functional and performance requirements.

Step 3. Identify the data items.

Step 4. Separate the data members from the persistent classes.

Step 5. Build a data member dictionary.

Step 6. Gather data members into persistent classes.

Step 7. Identify the key data members of each class.

Step 8. Identify the relationships between classes.

Step 9. Identify the class methods.

Step 10. Add the inheritance and member functions to make the classes persistent.

Figure 4.1 Ten Basic Object Database Design Steps

NPSNET_V traces its development through guidelines of ten basic object database design steps, shown in Figure 4.1 [Stev92]. An unlisted but important final step is refinement, particularly since overall NPSNET development is

ongoing. Highlights of these design steps, with respect to NPSNET_V, follow.

1. Identify the Basis

The basis for NPSNET_V rests in the mission and purpose of the NPSNET vehicle database. It reviews resources available for NPSNET_V development and proposes possible approaches to problem solution.

The mission of NPSNET_V is to update the existing NPSNET storage and retrieval facility. Its purpose is to support a real-time vehicle combat simulation that encompasses dynamic terrain and ballistic motion issues, intelligent autonomous agents, and three-dimensional representation, with expansion capability for future projects.

Resources available for NPSNET_V development include the present NPSNET database system and the parameter goals under which the simulator operates, as well as the objects that NPSNET models, primarily vehicles, their ordnance, and the terrain and obstacles which affect their movement. NPSNET progression toward object-oriented modeling and real-time scene management over a multiple workstation network makes object-oriented data management a likely approach to NPSNET vehicle data model design.

The NPSNET_V basis is an evolving description, changing as system updates occur. It is a necessary first step in data model design, laying the foundation for the

NPSNET_V requirements analysis. Figure 4.2 summarizes the NPSNET_V basis.

NPSNET_V Basis

Mission
Update NPSNET storage and retrieval facility.

Purpose
Support real-time vehicle combat simulation:
- Dynamic terrain
- Ballistic motion
- Intelligent autonomous agents
- Three-dimensional representation

Expansion capability.

Resources
- NPSNET database system
- Parameter goals of simulator operation
- Objects that NPSNET models:
  -- vehicles
  -- ordnance
  -- terrain
  -- obstacles

Possible solution
Object-oriented data management

Figure 4.2 Object Database Design Step 1: Identify the Basis

2. Define the Requirements

A requirements definition should state system needs, indicating what is to be done without specifying how it is to be done [Rumb91]. This step of the NPSNET_V database design process reviews both functional requirements and performance requirements.

NPSNET_V functional requirements detail the kind of data the vehicle database contains as well as the expected output of the system. They identify pieces of information that the database must comprise. NPSNET_V performance requirements include criteria about how the system will run, backup, recover and restart, as well as details about system

maintenance and database administration [Stev92]. Figure 4.3 summarizes these requirements.

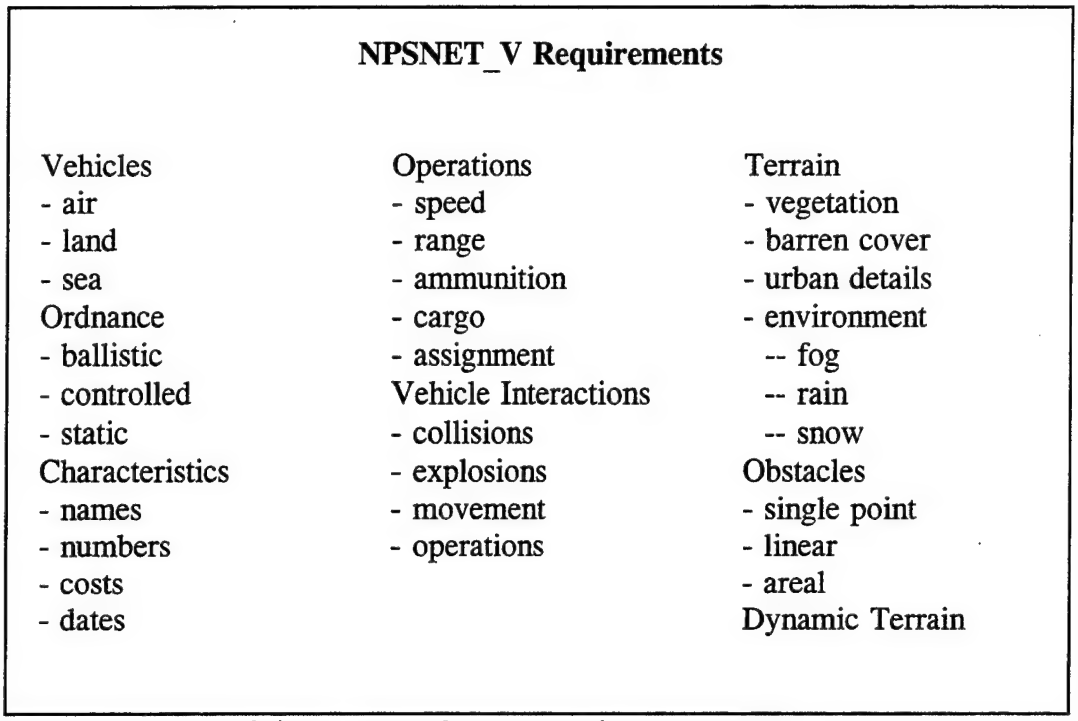


Figure 4.3 Object Database Design Step 2: Define the Requirements

NPSNET catalogues multiple air, land and sea vehicles, as well as ballistic, controlled and static ordnance. Its database must document their individual characteristics, such as names, numbers, costs and dates, as well as operational features like speed and range capabilities, and ammunition and cargo assignment.

The system tracks vehicle interactions, including collisions, explosions, movement and other operations. NPSNET also catalogues terrain and environmental conditions through which the vehicles and ordnance move, and obstacles that impact their movement. These include vegetation,

barren cover, urban details, fog, rain, snow, and single point, linear and areal fixed objects.

Ideally, the level of system detail will incorporate real-time scene management concerns, particularly those of concurrency control, point of reference, current object status, dynamic terrain features and vehicle state-of-readiness at a given point in the combat simulation. The functional and performance requirements will help identify data items that NPSNET_V must support.

3. Identify the Data Items

A simple and effective way to identify NPSNET_V data items is to extract probable data references, particularly nouns, from the NPSNET_V basis and requirements of the first two design steps. Each significant noun, or data item, may result in NPSNET_V class or data member representation in the next part of the object database design process.

Other data associated with these references will expand the list and assist in identifying additional data items. Verbs associated with the list are important for later use in identifying class methods. Organize the NPSNET_V data items in a fashion that allows for sorting and reordering. This will facilitate additional refinement, as the database design progresses.

Figure 4.4 presents an initial list of nouns from the basis and requirements steps. These data items may result in NPSNET_V class or data member representation.

NPSNET_V Data Items

Basis nouns:

dynamic terrain	vehicles
ballistic motion	ordnance
three-dimensional representation	terrain
	obstacles

Requirements nouns:

air vehicles	operations	terrain
land vehicles	speed	vegetation
sea vehicles	range	barren cover
ballistic ordnance	ammunition	urban details
controlled ordnance	cargo	environment
static ordnance	assignment	fog
characteristics	vehicle interactions	rain
names	collisions	snow
numbers	explosions	single point obstacles
costs	movement	linear obstacles
dates	dynamic terrain	areal obstacles

Figure 4.4 Object Database Design Step 3: Identify the Data Items

4. Separate the Data Members from the Classes

Data aggregates representing NPSNET_V persistent classes will come from the list of data items identified in the step above. Data members within these classes will derive from the other data items on the same list. The refined data item list of Figure 4.5 helps to highlight which items are data members and which are not. It is a reasonable place from which to start in the subjective separation of data members from persistent classes.

NPSNET_V Data Items

- **ballistic motion**
  - movement
  - 3-D representation
- **characteristics**
  - ammunition
  - cargo
  - costs
    - unit
  - crew
    - capacity
    - complement
  - dates
    - commission
    - decommission
  - names
    - crew member
    - hull
    - model
  - numbers
    - buno
    - hull
    - serial
    - social security
- **collisions**
  - vehicle with obstacle
  - vehicle with ordnance
  - vehicle with terrain
  - vehicle with vehicle
- **dynamic terrain**
  - collisions
  - explosions
  - terrain
- **explosions**
  - ordnance on obstacle
  - ordnance on ordnance
  - ordnance on terrain
  - ordnance on vehicle
- **movement**
  - acceleration
  - attitude
  - distance
  - direction
  - elevation
  - gridpoint
  - gridsquare
  - velocity
- **obstacles**
  - areal
    - city
    - crater
  - linear
    - barrier
    - dock
    - transport
  - point
    - antenna
    - building
    - buoy
- **operations**
  - assignment
  - consumption
    - rate
    - source
  - movement
  - platform
  - range
  - speed
  - storage
    - capacity
    - location
  - transfer
    - destination
    - source
- **ordnance**
  - ballistic
    - gun
    - howitzer
    - mortar
  - controlled
    - missile
    - torpedo
  - static
    - mine
- **terrain**
  - barren
    - dirt
    - mud
    - rock
    - sand
  - environment
    - fog
    - haze
    - rain
    - snow
- **terrain** (-continued)
  - urban
    - asphalt
    - city
    - railway
    - roadway
    - waterway
  - vegetation
    - brush
    - crop
    - forest
  - water
    - flowing
      - river
      - waterway
    - standing
      - lake
      - ocean
      - pond
      - puddle
      - swamp
- **3-D representation**
  - graphic
  - sound
- **vehicles**
  - air
    - aircraft
      - fixed wing
      - rotary
    - hydrofoil
  - land
    - amphibian
    - railed
    - tracked
    - wheeled
  - sea
    - submarine
    - surface
      - amphibian
      - hydrofoil
      - ship
- **vehicle interactions**
  - collisions
  - explosions
  - movement
  - operations

Figure 4.5 Refined List of NPSNET_V Data Items

Stevens cautions that the clear separation in traditional database design between aggregates and data elements does not exist in an object-oriented design. Here, some of the apparent data elements will in turn become abstract data types, or classes [Stev92]. This step must incorporate author judgment and discernment to differentiate between the two. One method of discernment is to separate the line entries of the data item list, rearranging and regrouping them to aid in data member recognition. Some items will stand out as singular data members, while others seem to naturally suggest aggregates of data members.

NPSNET_V Data Members

crew capacity	hull number	gridpoint
crew complement	serial number	gridsquare
unit cost	social security number	velocity
commission date	acceleration	consumption rate
decommission date	attitude	range
crew member name	distance	speed
hull name	direction	storage capacity
model name	elevation	storage location
buno number		

Additionally, each individual *instance* of:

collisions	obstacles	terrain
explosions	ordnance	3-D representation
		vehicles

Figure 4.6 Object Database Design Step 4: Separate the Data Members from the Classes

Figure 4.6 is an initial list of NPSNET_V data members, taken from the data item nouns of the previous

step. Separation of these data members from the persistent classes is necessary in preparation for the next two NPSNET_V design steps.

5. Build Data Member Dictionary

This design step ushers in the first construction phase of the NPSNET_V database solution. It builds a data member dictionary from the data members in NPSNET_V that will be members of persistent classes. The ability to organize, sort and rearrange this data remains important. Itemize each data member separately to aid in later identification of their redundancies and interclass relationships.

Building a data member dictionary requires comprehensive knowledge about its data members, including, at a minimum, the data type of each data member. For some items in Figure 4.6, data type determination is easy. Some of these data members will be instances of abstract data types, possibly taken from class libraries. For example, costs may be instances of a currency class, dates will be instances of a date class, and names will be instances of a string class [Stev92].

Dates and social security numbers are data members with known ranges and formats. Not all numbers are simple integer types, however. Some may be abstract integer types, with defined ranges, while others, like serial numbers, may contain letters or punctuation, as well as digits. The data

members that represent certain values may be quantities or amounts that must have defined limits, and those that refer to location may require a set of enumerated values.

Data typing is often application dependent. The format of a collision, explosion, obstacle, ordnance, terrain, three-dimensional, or vehicle data item, for example, is not obvious from the data member list alone. Because NPSNET is an existing system, its available source code and documentation can contribute to the construction of the data member dictionary. Stevens suggests using source code, in lieu of inadequate documentation, to first examine how existing software uses data members, and then to "reverse-engineer" their characteristics [Stev92].

Once each data member has a clearly defined format and behavior, construction can begin of classes that will implement these members. A clear and comprehensive definition of the physical properties of all data members within the database is an imminently important stage of NPSNET_V design, and should be a matter for future consideration. An in depth definition of these data members goes beyond the scope of this endeavor. The following steps use sample definitions for a representative selection of data members.

6. Gather Data Members into Classes

After separating the data members from the NPSNET_V data item list, aggregates remain. Most of these aggregates

NPSNET_V Classes

ballistic motion	operations	roadway
characteristics	assignment	waterway
ammunition	consumption	vegetation
cargo	consumption source	brush
costs	platform	crop
crew	storage	forest
dates	transfer	water
names	transfer destination	flowing
numbers	transfer source	river
dynamic terrain	ordnance	standing
collisions	ballistic	lake
vehicle with obstacle	gun	ocean
vehicle with ordnance	howitzer	pond
vehicle with terrain	mortar	puddle
vehicle with vehicle	controlled	swamp
explosions	missile	3-D representation
ordnance on obstacle	torpedo	3-D graphic
ordnance on ordnance	static	3-D sound
ordnance on terrain	mine	vehicles
ordnance on vehicle	terrain	air
movement	barren	aircraft
obstacles	dirt	fixed wing
areal	mud	rotary
city	rock	hydrofoil
crater	sand	land
linear	environment	amphibian
barrier	fog	railed
dock	haze	tracked
transport	rain	wheeled
point	snow	sea
antenna	urban	submarine
building	asphalt	surface
buoy	railway	ship
		vehicle interactions

Figure 4.7 Object Database Design Step 6: Gather Data Members into Classes

will become the persistent classes of NPSNET_V. Balance the remaining aggregates, listed in Figure 4.7, against the

NPSNET requirements to ensure that those left represent necessary persistent classes. This step is not absolute, as refinement will play a large role in the final determination of what are, and are not, persistent classes.

The intent of this step is to gather data members into the remaining NPSNET_V persistent classes by defining the data representation of each class. Identify the data members of each class. Gather together all data items related to the class, and build a figurative stack of the relevant data members. While this task sounds simple, it is often confusing and may require a great deal of refinement. Again, organize the information in a manner that allows for sorting and rearranging.

Next, identify the data types of each data member within the class, using the definitions of the data dictionary from the previous step. Once these definitions are in place and the necessary abstract data types exist, the design of each persistent class falls into place.

The four primary NPSNET_V object classes are the ordnance, vehicle, obstacle and terrain classes. Each is a superclass, representing an aggregate of several other, derived classes. Identification of an object within such a class requires representation of its class hierarchy, and subsequent data representation of each subclass. Extensive class hierarchies of the ordnance, vehicle, obstacle and terrain superclasses appear in the Appendix.

Consider data representation of a Mk 46 torpedo. The NPSNET_V Ordnance Class Hierarchy, found in the Appendix, lists this ordnance object as a member of the **MK_46** class of torpedoes. Figure 4.8 shows an example of a data representation of the **MK_46** class design, utilizing string, real number, character, simple integer, date and currency data member definitions.

```
class MK_46 {                          //Mk 46 torpedo
        String serialno;               //serial identification number
        String name;                   //"Mk 46 torpedo"
        String model;                  //"Mk 46"
        Real length;                   // 8.5  ft
        Real diameter;                 //12.75 ft
        Real weight;                   //508 lb
        Real max_speed;                //45 knots
        Real max_range;                //12,400 yd at optimum depth
        Real acquisition_range;        //more than 1,000 yd
        char piston_engine;            //piston engine (solid propel), yes/no
        char cam_engine;               //cam engine (liquid propel), yes/no
        int PBXN_103;                  //98 lb PBXN-103 high explosive
        Date date_prod;                //service entry date
        Currency cost_unit;            //unit cost
        String contractor;             //Honeywell
        String buno;                   //buno number of deployment helo
        String hullno;                 //hull number of deployment ship
};
```

Figure 4.8 The **MK_46** Class

Class design is an ongoing process. An existing class design should neither restrict a following design nor remain exempt from future review and revision. With this in mind, consider data representation of an SH-60B Seahawk helicopter. The **SH_60B** class is listed in the NPSNET_V Vehicle Class Hierarchy, also found in the Appendix. It derives from the "H-60 single rotary aircraft" path of the hierarchy. Figure 4.9 shows a sample **SH_60B** class design.

```
class SH_60B {                          //SH-60B Seahawk helicopter
        String buno;                    //buno identification number
        String name;                    //"Seahawk"
        String model;                   //"SH_60B"
        Real fuse_length;               //50 ft
        Real over_length;               //64.6 ft
        Real rotor_dim;                 //53.5 ft
        Real height;                    //17 ft
        Real max_gross_weight;          //21,884 lb
        int T700_GE_401C;               //twin T700-GE-401C Turboshaft engs
        int 1900_shaft_hp;              //1,900 shaft horsepower per engine
        Real max_speed;                 //180 knots
        Real max_range;                 //About 380 nm
        Real max_fuel;                  //4,000 lb
        Real consumrate_fuel;           //1,000 lb/hour
        char LAMPS_Mk_III               //LAMPS Mk III Weapons System
        int Mk46_torp;                  //2-Mk 46 torpedoes
        int Penguin;                    //1 Penguin missile
        int crew_comp;                  //crew complement of 3, takes up to 5
        int off_comp;                   //officer complement of 2
        int enl_comp;                   //enlisted complement of 1
        char rotor_brake;               //rotor brake, on/off
        char blade_fold;                //blade fold, yes/no
        char tail_fold;                 //tail fold, yes/no
        Date date_comm;                 //service entry date
        Currency cost_unit;             //unit cost
        String contractor;              //Sikorsky
        String hullno;                  //hull number of deployment ship
};                                      //
```

Figure 4.9 The **SH_60B** Class

An *Arleigh Burke*-class AEGIS destroyer also appears in the Appendix, as a member of the **DDG_51** class of surface ships. An example of its class design is in Figure 4.10.

Note that instances of both the **SH_60B** and **MK_46** classes may be assigned to the **DDG_51** class, and that both an **SH_60B** and a **DDG_51** may have assigned instances of the **MK_46** class. The requirement to deploy ordnance on a vehicle, or embark one vehicle on another, addresses design decisions that surpass the flat file representations of Figures 4.8, 4.9 and 4.10. A relational model would

```
class DDG_51{                          //Arleigh Burke-class DDG
        String hullno;                 //hull identification number
        String name;                   //"Arleigh Burke"
        String model;                  //"DDG"
        Real over_length;              //509.5 ft loa
        Real beam;                     // 66.7 ft
        Real draft;                    // 30.5 ft (navigational)
        Real max_displacement;         //9,9195 tons (full load)
        int gas_turbine;               //4 gas turbines
        int 100k_shaft_hp;             //100,000 shaft hp, 2 shafts
        Real max_speed;                //31+ knots
        Real max_range;                //4,400+ nm at 20 knots
        Real max_fuel;                 //maximum fuel load
        Real consumrate_fuel;          //fuel consumption rate
        char Flight_IIA;               //FlightIIA design, yes/no
        int SH_60B;                    //SH-60B Seahawk, embarked
        int 64_Mk41_VLS;               //1 64-cell Mk 41 VLS
        int 32_Mk41_VLS;               //1 32-cell Mk 41 VLS
        int Harpoon;                   //Harpoon Anti-Ship Missile
        int SM_2;                      //Navy's Standard Missile SM-2
        int TLAM;                      //Tomahawk Land-Attack Missile
        int TASM;                      //Tomahawk Anti-Ship Missile
        int 12_75;                     //6 12.75-in torp tubes (2 trip mnts)
        int Mk46;                      //Mk 46 torpedo
        int Mk50;                      //Mk 50 torpedo
        int 5_54_Mk45;                 //1 5-in 54-cal Mk 45 dual purp gun
        int 20_Mk15;                   //2 20-mm Mk 15 Phalanx CIWS
        char AEGIS;                    //AEGIS Weapon System
        int Mk99;                      //3 Mk 99 illuminators
        char SPY_1D;                   //AN/SPY-1D multi-function radar
        char SPS_67V3;                 //AN/SPS-67(V)3 surface search radar
        char SPS_64;                   //AN/SPS-64 navigation radar
        char SQS_53C;                  //AN/SQS-53C bow-mounted sonar
        char Kingfisher;               //rev Kingfisher mine-detection system
        int crew_comp;                 //383 crew complement, with helo det
        int off_comp;                  // 32 officers complement
        int enl_comp;                  //351 enlisted complement
        Date date_comm;                //commissioned 12 Dec 92
        Currency cost;                 //unit cost
        String contractor;             //Bath Iron Works
};
```

Figure 4.10 The **DDG_51** Class

represent this relationship with a file that records the ordnance-vehicle or vehicle-vehicle assignment, normalizing the design to avoid inefficiency. The data representation presented in Figure 4.11 is an example of an **Assignment** class design.

```
class Assignment {                          //Assignment class
        String idno;                        //assigned object identification number
        String platformno;                  //assignment platform identification number
};
```

Figure 4.11 The **Assignment** Class

This **Assignment** class serves to broadly reflect all ordnance and vehicle assignments within NPSNET_V. The identification number represents Mk 46 serial numbers, as well as SH-60B buno numbers. The platform number represents both buno and hull numbers.

Some relationships between NPSNET_V classes are more complicated than others. Each data representation requires the designer to iterate the design process, revisit the requirements, and make appropriate modifications. Only a few examples of NPSNET_V class design appear in this work. Full data representation of all NPSNET object classes is left as a matter for future consideration. The ones listed in this section serve to illustrate the potential of NPSNET_V through the remaining object database design steps.

7. Identify the Key Members

Each NPSNET_V class must have a primary key data member to identify objects of the class. Each class may also have one or more secondary keys, which identify alternate ways to locate an object within the database.

In the class design examples of the previous step, the serial number is the primary key for the **MK_46** class, and it uniquely identifies each instance of a **MK_46** object.

Similarly, the buno number is the primary key for the **SH_60B** class and the hull number is the primary key for the **DDG_51** class. Each object has a unique identification number.

The hull number is a secondary key for both the **MK_46** class and the **SH_60B** class. It identifies a ship deployment platform for each torpedo or helicopter. The **MK_46** class also contains a buno number to identify a helicopter deployment platform, in lieu of a hull number. Every **MK_46** object has either a hull number or a buno number, and multiple Mk 46 torpedoes can have the same hull or buno number.

The **Assignment** class in NPSNET_V is similar to a connector file in a relational database. Its primary key is the concatenation of the identification number and the platform number. The combination of these two data members uniquely identifies each instance of an **Assignment** object.

Every object in NPSNET_V has some feature that uniquely identifies it. During class design, base the primary key of each class on the identification feature that sets each object apart from other members of its class. If an NPSNET_V persistent class has no primary key, then redesign that class.

Secondary keys provide alternate ways to locate objects of a persistent class. They imply a relationship between classes when the primary key of one class is the secondary key of another. In the primary key of a connector

class, each concatenated key member is a secondary key of the connector class, as well as the primary key of one of the connected classes. The next design step touches on the class relationships that secondary keys support.

8. Identify Class Relationships

To identify relationships between NPSNET_V persistent classes, identify those classes that share a common data member. If that data member is the primary key of one of the classes, then those classes are related.

Stevens cautions that an implied relationship must have integrity, in order for it to work [Stev92]. This means that if an object of a first class contains the primary key of a second class, then there should be a matching object of that second class. For instance, if a particular **MK_46** class torpedo is assigned to a **DDG_51** class ship as part of its armament, then there must exist a **DDG_51** class object with a hull number that exactly matches the hull number in the secondary key of that particular **MK_46**.

Relationships between classes are important because they serve as potential paths for multiple-class retrievals. For example, consider the relationships between the **MK_46**, **DDG_51**, and **SH_60B** classes. A path that first retrieves a **DDG_51** object, then each **SH_60B** deployed on that ship, and then each **MK_46** assigned to each of those helicopters, will return the Mk 46 torpedoes that are deployed on the SH-60B helicopters assigned to that DDG.

If the design intent were to recover a list of torpedoes deployed directly on a particular DDG, and not those assigned to a helicopter, then the return would be incorrect. Review class relationships carefully, and develop retrieval paths meticulously, to ensure that data returns accurately reflect the intent of the retrievals.

9. Identify the Methods

Identifying the NPSNET_V class methods entails a review of the design step that gathered these classes. Return to the list of potential persistent classes, and seek out verbs that may define application-dependent behavior of the persistent classes. Figure 4.12 is a brief list of verbs drawn directly from the nouns in the NPSNET_V Classes list of Figure 4.7.

NPSNET_V Class Methods

collide	operate	store	flow
explode	assign	transfer	snow
move	consume	rain	interact

Figure 4.12 Object Database Design Step 9: Identify the Methods

Expand on this list, organizing the information to allow for sorting and rearranging. Identify data member behaviors, and define these behaviors as object-oriented methods. Figure 4.13 features an expanded list of potential class methods, grouped to focus on similar behaviors.

NPSNET_V Class Methods

operate	interact	move	reverse	blow
assign	collide	accelerate	roll	flow
close	explode	ascend	sink	haze
consume	aim	decelerate	start	mist
open	fire	descend	stop	rain
store	load	float	turn	sleet
transfer	track	pitch	yaw	snow

Figure 4.13 Expanded list of NPSNET_V Class Methods

This portion of the design is similar to the design of any object-oriented program. Identify the NPSNET_V methods, then add them to the appropriate NPSNET_V persistent classes. Stevens confirms that each of these methods will bind to its persistent object because the program that retrieves the object uses the same class definition and class library used by the program that created the object [Stev92]. The remaining object database design step makes each NPSNET_V class persistent.

10. Make the Class Persistent

According to Stevens, the last step of object data model design, aside from revision, is integration of the persistent object class design into the persistent object database management system [Stev92]. Do this by adding attributes to the NPSNET_V key data members and persistent classes, enabling them to participate in their own persistence.

The type of object database management system that NPSNET employs will influence the nature of these attributes. Generally, Stevens suggests that these attributes will consist of inheritance, custom base class functions, and special member functions added to the persistent class that the base will call [Stev92].

Once its design is complete, the NPSNET_V data model will require a database management system to put it into effect. Selection of the system software, and implementation of such a system, are both projects for future research and development. The following section discusses issues affecting NPSNET_V implementation.

C. NPSNET_V: DATA MODEL IMPLEMENTATION

Implementing the NPSNET_V object data model will require the selection of a software system to process the database. PARODY is an example of a database management system that can define NPSNET_V, and bring it together with the NPSNET system. According to Stevens, PARODY is a non-proprietary "Persistent, Almost Relational Object Database" system that can be implemented as a C++ class library [Stev92].

PARODY is aptly named, for its data model is itself a type of parody, strongly resembling the traditional relational data model, while assuming certain properties of C++ objects. It can provide the persistent capabilities that NPSNET_V needs, but that object-oriented programming languages alone do not support.

1. Define NPSNET_V Database

A project for future consideration would be to define NPSNET_V as a PARODY database. Traditional database file definition involves building a data model with a data definition language to describe file content. PARODY could enable NPSNET_V to use C++ class definition. PARODY would encapsulate into the C++ class definition a description of the data members in each class, plus the integration of the class with the PARODY database manager.

NPSNET_V definition would first involve class designation to represent NPSNET_V persistent objects, already begun in the design steps of the previous section. It would next identify key data members for these classes. Abstract base classes already defined in the PARODY class library could facilitate NPSNET_V derivation of persistent object classes and key member classes.

Although an object-oriented programming language such as C++ cannot sustain object persistence on its own, PARODY could solve this problem for NPSNET_V. It would provide specific member functions to allow objects of persistent class types to participate in their own persistence.

Additional definition issues to consider would include relating NPSNET_V classes, limiting multiple-copy objects, and embedding persistent objects within other ones, to streamline application efficiency.

2. Bring NPSNET_V Together

A next step in NPSNET_V implementation could be to write the program that brings the NPSNET_V design together with the existing NPSNET system. This program would first build a database, based on the NPSNET_V object data model design. It would then declare and use NPSNET_V persistent objects to achieve NPSNET real-time vehicle combat simulation. This program would also have to be able to destroy NPSNET_V persistent objects. Finally, the program would have to tie NPSNET_V object input and output to NPSNET operations, guaranteeing system integrity along the way.

Implementation of the NPSNET_V design will require insightful planning and thoughtful programming in order to achieve an operational NPSNET object database management system. The effort will be validated by the increased real-time scene management capabilities of an enhanced NPSNET distributed network.

V. CONCLUSION

This work proposes NPSNET_V as a vehicle database model for NPSNET, a real-time vehicle and battle simulator. It considers the progression of NPSNET toward object-oriented modeling and real-time scene management over a distributed network, citing the need for a database with increased storage and retrieval capabilities. It reviews traditional and available data models, and documents the selection and design of the NPSNET_V object data model. In conclusion, this work discusses the viability of NPSNET_V as a data mechanism for use with NPSNET, highlighting recommendations for future work.

A. RESULTS

NPSNET_V is a viable object data model for use in upgrading the data management facility of the NPSNET vehicle combat simulator. It presents a probable approach to the problem of current NPSNET data storage and retrieval mechanisms which are slow, cumbersome to maintain and complex to enhance. As NPSNET proceeds in the direction of object-oriented modeling and real-time scene management over a multiple workstation network, its data management system must also progress.

NPSNET_V provides a design for an updated database facility to support the demands involved in NPSNET vehicle combat simulation. It adapts the relational data model to

represent NPSNET objects, combining its strengths with the functional capabilities of object-oriented design.

NPSNET_V can define related persistent classes and their behavior within NPSNET to handle the data management requirements associated with dynamic terrain, ballistic motion, intelligent autonomous agents, and three-dimensional representation. It can anticipate the needs of future projects which will require speed and accuracy in documenting world state events, with built-in concurrency control to manage user interface across a distributed network.

The proposed NPSNET_V design is a reasonable place from which to proceed with implementation of an updated NPSNET Vehicle Database. The following section recommends areas for future research and development.

B. RECOMMENDATIONS

This work identifies, develops and explores the design of a data model to enhance the existing NPSNET data management system. It also recommends areas for future consideration and implementation.

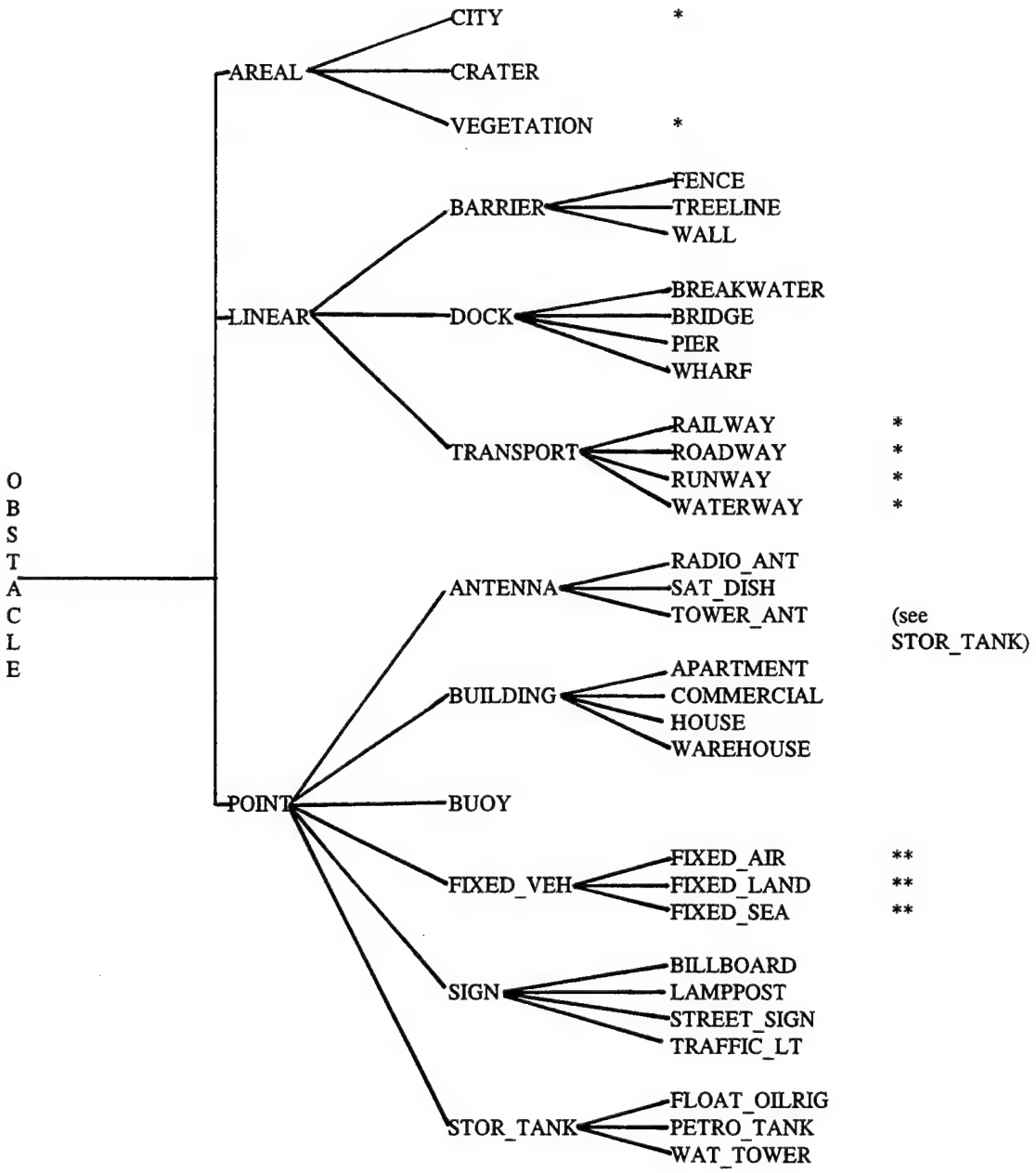
In particular, Chapter IV highlights the need for a clear and comprehensive definition of the physical properties of all data members within the proposed NPSNET database, as well as the need for full data representation of all NPSNET object classes.

This works also outlines the selection of database management system software, and implementation of such a system, as projects for future NPSNET research and development. Specifically, it recommends exploring NPSNET_V as a PARODY database, directly capitalizing on the design groundwork laid in Chapter IV.

Future design considerations should accommodate ongoing NPSNET simulator development. Data models continue to evolve to meet the needs of evolving programming languages, and NPSNET should capitalize on the merits of both.

APPENDIX

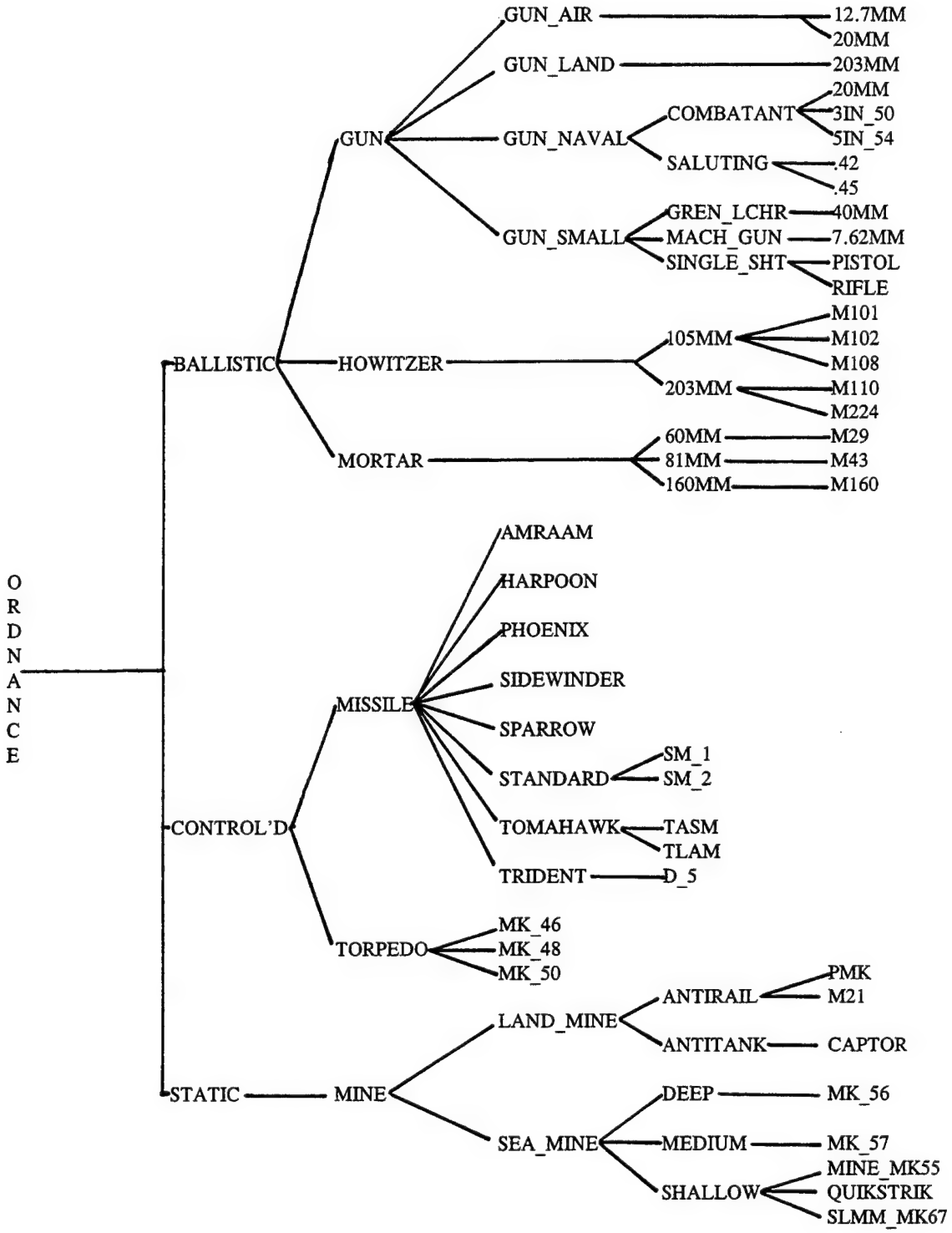
Obstacle Class Hierarchy



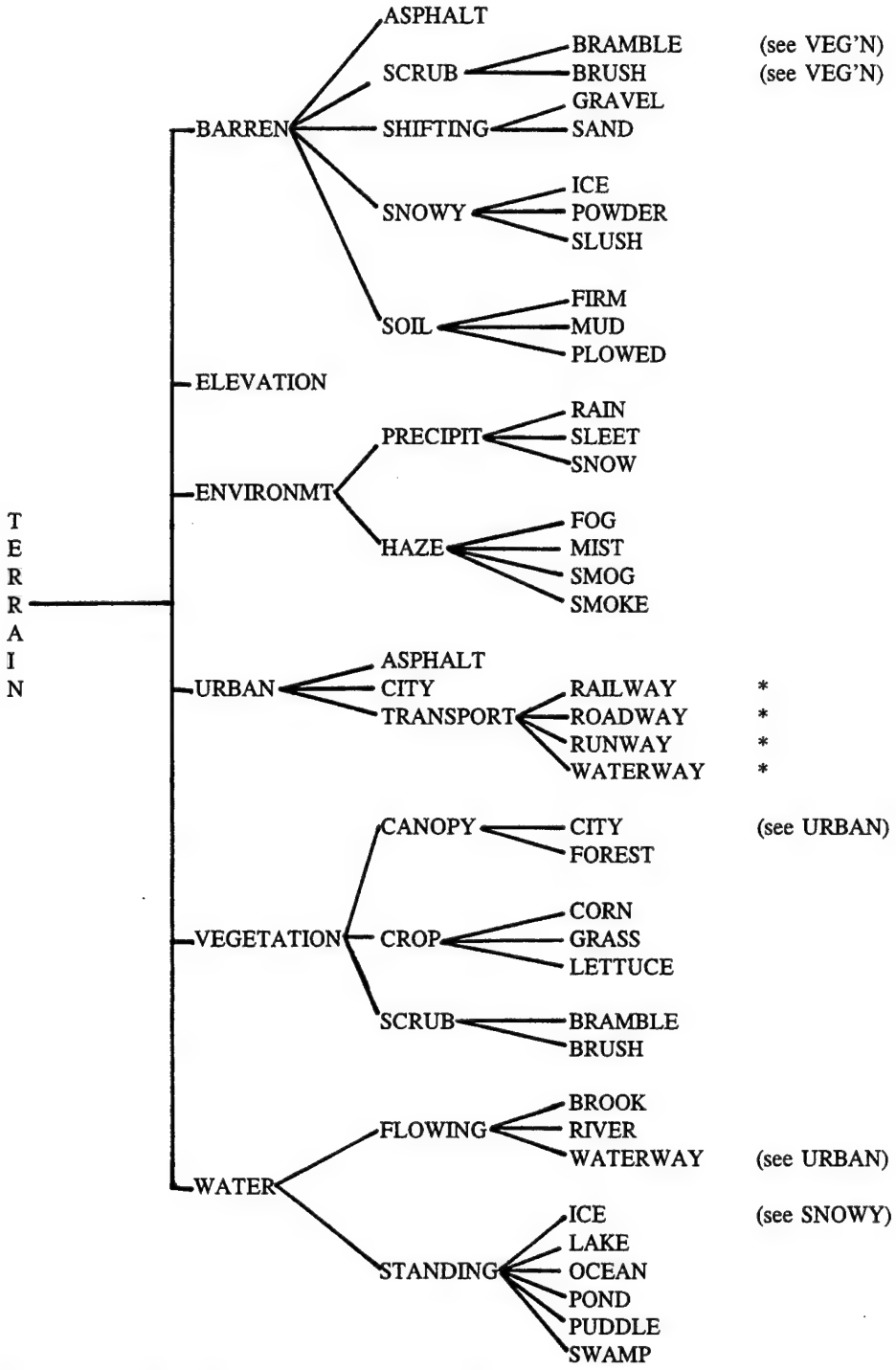
* **Possible case of multiple inheritance; see TERRAIN class.**

** **Possible case of multiple inheritance; see VEHICLE class.**

Ordnance Class Hierarchy

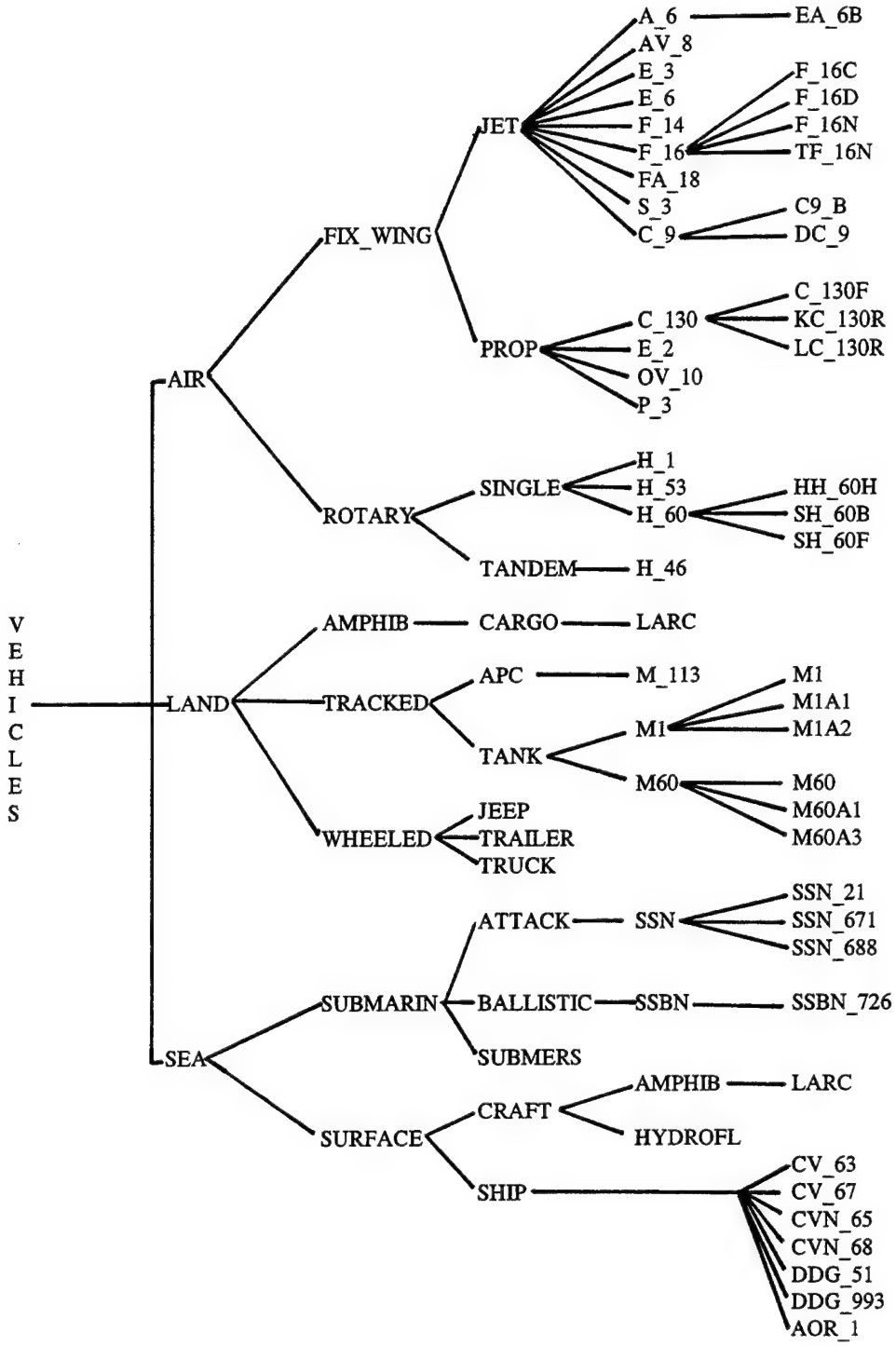


Terrain Class Hierarchy



* Possible case of multiple inheritance; see OBSTACLE class.

Vehicle Class Hierarchy



GLOSSARY

abstract data type: A user-oriented construct; a user defined data type that is not built into the programming language.

abstraction: Defining new data types; designing a class to define an abstract data type.

aggregate: A collection, as in an **aggregate** of data members.

base class: A class from which other classes are derived. The derived class inherits all of the characteristics of the base class. See *superclass*.

child record: A database record that is related to higher, or parent, records in the database.

class: A user-defined data type consisting of data members and member functions.

class hierarchy: : A system, or data lattice, of base and derived classes.

data language: A language used to write transactions which refer to the data stored in a database that is managed by a database management system. Also, a language that specifies the processing, integrity and security requirements in a database.

data member: A data component of an object-oriented class; any valid data type.

data model: A database design that enables the user to specify a database in terms of abstract and concrete features about types, aggregates, and relationships of data stored in the database.

database: Stored data; a unification of several otherwise distinct data files to support a common application.

database management: Storage and retrieval of data in a database.

database management system: The software that handles all storage and retrieval of data in a database.

derived class: A class that inherits all of the characteristics of the base class. See *subclass*.

element: A single piece of data; one item of a data type. Collections of them form files in a database. See *record*.

encapsulation: Defining data type representation and behavior in an encapsulated entity; defining a class by binding its data members and functions together. Encapsulation implies that the implementation is transparent to the user, while the interface is visible to the user.

file: A collection of records of a common database format.

hierarchical data model: A database design of parent and child records that emulates an inverted tree structure, in which each parent record may have multiple child records, but each child record may have only one parent record. A child record may have parent-to-lower-child records relationship, forming a multiple-level hierarchy.

inheritance: Deriving a new data type from an existing one; the ability for a subclass to inherit both data attributes and methods from previously defined objects in a nested or hierarchical fashion. Also referred to as subclassing.

instance: An object that has state, behavior and identity. Also, an object that has been described by a class; the object is called the **instance** of that class.

network data model: A database design of parent and child records that emulates a lattice structure in which each parent record may have multiple child records and each child record may have multiple parent records. A child record may have a parent-to-lower-child records relationship, forming a multiple level network.

object: An instance of a data type, including standard object-oriented programming language data types as well as objects of classes.

object data model: A database design of a collection of persistent objects.

object-oriented: Containing the properties of abstraction, encapsulation, inheritance and polymorphism; also, defining abstract data types, instantiating objects, and sending messages to the object's methods.

object-oriented database management system: The software that allows the user to use and maintain the objects in an object-oriented database.

parent record: a database record that is related to lower, or child, records in the database.

persistence: The ability of an object to succeed its creator and to subsequently exist in space other than the space in which it was created. Also, the property of being written to storage and then retrieved.

pointer: A variable used to hold values that are the addresses of objects in memory. A pointer references an object indirectly, and can manage objects allocated during program execution.

polymorphism: Customizing the behavior of a derived data type; exhibition by the methods in a class hierarchy of different behavior for the same message, depending on the type of the object for which the method is invoked, and without regard to the class type of the object reference.

primary key: A key data field whose value uniquely identifies a given record in a database file.

record: A group of related data elements that form to support one functional aspect of a database's application. A collection of records of common format is a *file*.

relational data model: A database design that represents data as tables of rows and columns. Any relationships between tables are formed by common values in common columns.

relationship: An association that links data records together.

secondary key: a key data value that indexes a file on other than its primary key. The value of a secondary key does not have to be unique. Multiple records can have the same secondary key value. When a secondary key is the primary key of another file, the two files have a many-to-one relationship.

subclass: See *derived class*.

superclass: See *base class*.

type: Enumeration of a data member in the database. In object-oriented design, refers to the **type** of a program constant or variable, which can be of a primitive or an abstract data type.

REFERENCES

[Bert91] E. Bertino and L. Martino. "Object-Oriented Database Management Systems: Concepts and Issues," *IEEE Computer*, Apr 1991, pp 33-47.

[Booc91] G. Booch. *Object-Oriented Design: With Applications*. Benjamin/Cummings Publlishing Co, Redwood City, CA, 1991.

[Date81] C. J. Date. *An Introduction to Database Systems*. Addison-Wesley Publishing Co, Reading, MA, 1981.

[dePa91] E. G. de Paula and M. L. Nelson. *Clustering, Concurrency Control, Crash Recovery, Garbage Collection, and Security in Object-Oriented Database Management Systems*. Naval Postgraduate School, Monterey, CA, Feb 1991.

[Elma89] R. Elmasri and S. B. Navathe. *Fundamentals of Database Systems*. Benjamin/Cummings Publishing Co, Redwood City, CA, 1989.

[Higa92] K. Higa, M. Morrison, J. Morrison and O. Sheng. "An Object-Oriented Methodology for Knowledge Base/Database Coupling," *Communications of the ACM*, Vol 36, No 6, Jun 1992, pp 99-101.

[Hsia91] D. K. Hsiao. "The Object-Oriented Database Management: A Tutorial on its Fundamentals," Naval Postgraduate School, Monterey, CA, Aug 1991.

[Hsia92] D. K. Hsiao. "Federated Databases and Systems: Part I - A Tutorial on Their Data Sharing," *VLDB Journal*, Vol 1, Aug 1992, pp 127-179.

[Osbo91] W. D. Osborne. *NPSNET: An Accurate Low-Cost Technique for Real-Time Display of Transient Events: Vehicle Collisions, Explosions and Terrain Modifications*. M.S. Thesis, Naval Postgraduate School, Monterey, CA, Sep 1991.

[Stev92] A. Stevens. *C++ Database Development*. MIS:Press, New York, NY, 1992.

[Zyda92] M. J. Zyda and D. R. Pratt. "NPSNET Digest: A Look at a 3D Visual Simulator for Virtual World Exploration and Experimentation," Naval Postgraduate School, Monterey, CA, Jun 1992.

[Zyd92] M. J. Zyda, D. R. Pratt, J. G. Monahan and K. P. Wilson. "NPSNET: Constructing a 3D Virtual World," *Proceedings of the 1992 Symposium on Interactive 3D Graphics (in cooperation with ACM SIGGRAPH)*, Cambridge, MA, Mar-Apr 1992, pp 147-156.

[Zyda93] M. J. Zyda, D. R. Pratt and K. M. Kelleher. "1992 NPSNET Research Group Overview," Naval Postgraduate School, Monterey, CA, Mar 1993.

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center — 2
   8725 John J. Kingman Road, Suite 0944
   Fort Belvoir, Virginia 22060-6218

2. Dudley Knox Library — 2
   Naval Postgraduate School
   411 Dyer Road
   Monterey, California 93943-5101

3. Electronic Systems Center — 2
   66 SPTG/SCXI
   50 Griffiss Street
   Hanscom Air Force Base, MA 01731-1621

4. LCDR Susan C. Borden Davis — 2
   Naval War College (CNC&S)
   Newport, RI 02841

5. Mr. Lawrence L. Borden — 1
   39 Mountain Drive
   Pittsfield, MA 01201